On the *QR* Algorithm and Updating the SVD and the *URV* Decomposition in Parallel

Marc Moonen*
ESAT Katholieke Universiteit Leuven
K. Mercierlaan 94
3001 Heverlee, Belgium

Paul Van Dooren†
University of Illinois at Urbana-Champaign
1308 West Main Street
Urbana, Illinois 61801

and

Filiep Vanpoucke‡
ESAT Katholieke Universiteit Leuven
K. Mercierlaan 94
3001 Heverlee, Belgium

Submitted by Gregory S. Ammar

ABSTRACT

A Jacobi-type updating algorithm for the SVD or the *URV* decomposition is developed, which is related to the *QR* algorithm for the symmetric eigenvalue problem. The algorithm employs one-sided transformations, and therefore provides a cheap alternative to earlier-developed updating algorithms based on two-sided transformations. The present algorithm as well as the corresponding systolic implementation is therefore roughly twice as fast as the former method, while the tracking properties are preserved. The algorithm is also extended to the two-matrix *Q*SVD or

* E-mail: moonen@esat.kuleuven.ac.be.
† E-mail: vdooren@uicsl.csl.uiuc.edu. Supported by DARPA under grant 60NANB2D1272
‡ E-mail: vpoucke@esat.kuleuven.ac.be.

655 Avenue of the Americas, New York, NY 10010 0024-3795/93/$6.00

QURV case. Finally, the differences with a number of closely related algorithms that have recently been proposed are discussed.

I. INTRODUCTION

In an earlier report [16], an adaptive algorithm has been developed for tracking the singular-value decomposition of a data matrix when new observations (rows) are added continuously. The algorithm may be organized so that it provides at each time a certain approximation for the exact singular-value decomposition. It combines a Jacobi-type diagonalization scheme, based on two-sided orthogonal transformations [12], with *QR* updates. A systolic implementation for this algorithm is described in [17].

Here, we improve upon these results. An alternative algorithm is described, which employs only one-sided transformations. Row and column transformations are applied in an alternating fashion. The algorithm is therefore roughly twice as fast, whereas its tracking properties are the same as for the two-sided method. The corresponding systolic implementation is roughly the same, but also twice as fast.

The algorithmic development starts from a square-root version of the *QR* algorithm for the symmetric eigenvalue problem [7, 15], in Section III. This algorithm is turned into a Jacobi-type algorithm, based on 2×2 transformations, by supplementing it with a permutation scheme, in Section IV. The resulting algorithm may then be interlaced with a *QR* update, whenever a new row has to be worked in, so that an adaptive scheme is obtained, in Section V. As the algorithm is operated without shifts of the origin, it is particularly suitable for isolating a cluster of small singular values. This is precisely the aim of a *URV* decomposition [21]. Thus, at each time, either an exact SVD may be computed or a *URV*-type approximate decomposition. In Section VI, the algorithm is extended to the 2-matrix QSVD or *QURV* case. In contrast to the *QURV* method proposed in [4], no preliminary extraction of a triangular factor is performed here. Moreover, a systolic array implementation is simpler with the present version. Some of these differences are explained in the last section, on related work by others.

II. PRELIMINARIES

The starting point here is a real[1] data matrix A, which is assumed to be tall and thin, i.e. with more rows than columns. The aim is to compute its

[1] We consider real arithmetic, for simplicity. The complex case is similar.

singular-value decomposition

$$\underbrace{A}_{N \times m} = \underbrace{U}_{N \times m} \cdot \underbrace{\Sigma}_{m \times m} \cdot \underbrace{V^T}_{m \times m}$$

with $U^T U = V^T V = I$ and Σ a diagonal matrix. In real-time applications, A is defined in a recursive manner, i.e., A at time k equals A at time $k-1$ plus one additional new observation (row):

$$A_k = \left[\begin{array}{c} A_{k-1} \\ \hline a_k^T \end{array}\right].$$

Mostly, exponential forgetting is applied with a forget factor $\lambda < 1$, i.e.

$$A_k = \left[\begin{array}{c} \lambda A_{k-1} \\ \hline a_k^T \end{array}\right].$$

Very often, the SVD is used for so-called *subspace tracking* applications. The matrix A_k is then supposed to have a clear gap in the singular-value spectrum. The larger singular values correspond to the *signal subspace*; the smaller singular values correspond to the *noise subspace*. The SVD may be written as

$$A = \underbrace{\left[\begin{array}{c|c} U_s & U_n \end{array}\right]}_{U} \cdot \underbrace{\left[\begin{array}{c|c} \Sigma_s & \\ \hline & \Sigma_n \end{array}\right]}_{\Sigma} \cdot \underbrace{\left[\begin{array}{c} V_s^T \\ \hline V_n^T \end{array}\right]}_{V^T},$$

where Σ_s contains the larger *signal singular values*, and Σ_n contains the smaller *noise singular values*. The aim is not to compute the complete singular-value decomposition, but rather to compute a good estimate for the subspaces V_s^T and V_n^T. Therefore, it is not necessary to have an exact diagonal matrix in the middle. An *approximate decomposition*, with e.g. a triangular matrix R in the middle,

$$A = \underbrace{\left[\begin{array}{c|c} \tilde{U}_s & \tilde{U}_n \end{array}\right]}_{\tilde{U}} \cdot \underbrace{\left[\begin{array}{c|c} R_s & R_{sn} \\ \hline & R_n \end{array}\right]}_{R} \cdot \underbrace{\left[\begin{array}{c} \tilde{V}_s^T \\ \hline \tilde{V}_n^T \end{array}\right]}_{\tilde{V}^T},$$

also yields good estimates for the subspaces, as long as the "cross term" $\|R_{sn}\|_F$ is small, so that R_s (R_n) has roughly the same singular values as Σ_s (Σ_n). In [16], this is called (somewhat loosely) an "approximate SVD," whereas in [21] it is called a "*URV* decomposition," referring to the separate factors.

In subspace tracking applications, the aim is to have a good estimate for the subspace V_s^T or V_n^T at each time instant k. The *tracking error*, which may be defined in terms of the angles between the exact and approximate subspaces V_s^T and $\tilde{V}_s^T$ [16], should be small at all times. Our aim is to develop efficient adaptive and parallel algorithms for this.

III. SQUARE-ROOT *QR*

In this section, we focus on computing the SVD of a fixed matrix A. It is shown how a Jacobi-type algorithm may be derived from the *QR* algorithm for the symmetric eigenvalue problem.

The SVD of the matrix A may be computed in two steps. First, a *QR* decomposition is computed, resulting in

$$\underbrace{A}_{N \times m} = \underbrace{Q_A}_{N \times m} \cdot \underbrace{R_A}_{m \times m},$$

where $Q_A^T Q_A = I$ and R_A is upper triangular. This is done in a finite number of time steps, e.g. with a sequence of Givens transformations [11]. Then an iterative procedure is applied to the triangular factor R_A, transforming it into a diagonal matrix. This diagonalization procedure consists in applying a sequence of plane transformations as follows (see [12, 13] for details):

$R \Leftarrow R_A$,
$\overline{U} \Leftarrow I$,
$\overline{V} \Leftarrow I$;

FOR $k = 1, \ldots, \infty$
  FOR $i = 1, \ldots, m-1$
    $R \Leftarrow \overline{U}_{[i,k]}^T \cdot R \cdot \overline{V}_{[i,k]}$
    $\overline{U} \Leftarrow \overline{U} \cdot \overline{U}_{[i,k]}$
    $\overline{V} \Leftarrow \overline{V} \cdot \overline{V}_{[i,k]}$
  END
END

The parameter i is called the *pivot index*. The matrices $\bar{U}_{[i,k]}$ and $\bar{V}_{[i,k]}$ represent orthogonal transformations in the $\{i, i+1\}$ plane. $\bar{U}_{[i,k]}$ differs from the identity only in the entries $(i,i) = (i+1,\ i+1) = \cos\theta$, $(i, i+1) = \sin\theta$, and $(i+1, i) = -\sin\theta$. Similarly, $\bar{V}_{[i,k]}$ differs from the identity only in the entries $(i,i) = (i+1, i+1) = \cos\phi$, $(i, i+1) = \sin\phi$, and $(i+1, i) = -\sin\phi$. The angles ϕ and θ are chosen so that applying $\bar{U}_{[i,k]}$ and $\bar{V}_{[i,k]}$ to R results in a zero $(i, i+1)$ entry in R, while R still remains in upper triangular form. Between the two possible solutions one chooses the so-called outer rotations closest to a 2×2 permutation [13]. Each iteration thus essentially reduces to performing a particular 2×2 SVD on the main diagonal. At each stage we have

$$R_A = \bar{U} \cdot R \cdot \bar{V}^T.$$

Furthermore, each rotation reduces the norm of the off-diagonal part in R. Under mild conditions [8], one shows that the off-diagonal part converges to zero and hence R converges to a diagonal matrix, resulting in the required SVD:

$$\begin{aligned} A &= Q_A \cdot R_A \\ &= \underbrace{Q_A \cdot \bar{U}}_{U} \cdot \underbrace{R}_{\Sigma} \cdot \underbrace{\bar{V}^T}_{V^T}. \end{aligned}$$

If these conditions are not satisfied, one easily modifies the standard algorithm to ensure global convergence to a diagonal matrix [8]. This SVD algorithm is simple, is amenable to parallel implementation [13], and may be turned into an adaptive algorithm [16].

In a way, the above algorithm may be viewed as a *square-root* version of the original Jacobi algorithm for the symmetric eigenvalue problem, applied to $A^TA = R^TR$ [11]. What is remarkable now is that another popular algorithm for the symmetric eigenvalue problem, namely the QR algorithm, may be turned into a Jacobi-type square-root algorithm, too.

The original QR algorithm, applied to A^TA, works as follows [11]:

$$X_0 \Leftarrow A^TA;$$

FOR $k = 0, \ldots, \infty$

$$\begin{bmatrix} \mathcal{Q}_k \cdot \mathcal{R}_k = X_k \\ X_{k+1} \Leftarrow \mathcal{R}_k \cdot \mathcal{Q}_k \end{bmatrix}$$

END

In each iteration, a QR factorization of X_k is computed. Then the next iterate X_{k+1} is obtained by reversing the order of $\mathscr{Q}_k$ and $\mathscr{R}_k$ and carrying out the multiplication. It is proved that—except for contrived examples—X_k converges to a diagonal matrix with the eigenvalues of A^TA ordered along the diagonal, i.e. $X_\infty = \Sigma^2$.

A square-root version of this algorithm has been derived in [7, 5, 9, 15] (see also [6] for a related result). With $A = Q_A R_A$, one has

$$X_0 = \underbrace{R_A^T}_{\text{lower}} \cdot \underbrace{R_A}_{\text{upper}} \overset{\text{def}}{=} R_0^T \cdot R_0.$$

The QR factorization of X_0 (cf. the first iteration) is obtained from the QR factorization of R_0^T:

$$X_0 = \overbrace{Q_{0\sharp} R_{0\sharp}}^{R_0^T} \cdot R_0$$

$$= \underbrace{Q_{0\sharp}}_{\mathscr{Q}_0} \cdot \underbrace{R_{0\sharp} \cdot R_0}_{\mathscr{R}_0}.$$

The next iterate is then obtained as

$$X_1 = \mathscr{R}_0 \cdot \mathscr{Q}_0$$

$$= R_{0\sharp} \cdot R_0 \cdot Q_{0\sharp}$$

$$= \underbrace{R_{0\sharp}}_{\text{upper}} \cdot \underbrace{R_{0\sharp}^T}_{\text{lower}}.$$

Finally, the original factorization (lower times upper) is restored, by computing and substituting the QR factorization of $R_{0\sharp}^T$:

$$X_1 = \overbrace{R_{0\dagger}^T Q_{0\dagger}^T}^{R_{0\sharp}} \cdot \overbrace{Q_{0\dagger} R_{0\dagger}}^{R_{0\sharp}^T}$$

$$= R_{0\dagger}^T \cdot R_{0\dagger}$$

$$\overset{\text{def}}{=} R_1^T \cdot R_1.$$

The above process is then repeated to compute similar factorizations for X_2, X_3, etc. One can verify that, with tidied-up notation,[2] a square-root algorithm is obtained as follows:

$$R_0 \Leftarrow R_A;$$

FOR $k = 0, \ldots, \infty$

$$\begin{bmatrix} \underbrace{L_k}_{\text{lower}} & \Leftarrow & \underbrace{R_k}_{\text{upper}} & \cdot & \underbrace{Q_{k\sharp}}_{\text{orthogonal}} \\ \underbrace{R_{k+1}}_{\text{upper}} & \Leftarrow & \underbrace{Q_{k\dagger}^T}_{\text{orthogonal}} & \cdot & \underbrace{L_k}_{\text{lower}} \end{bmatrix}$$

END

It is seen that column and row transformations are applied in an alternating fashion, wherewith an upper triangular factor is turned into a lower triangular factor and vice versa. Here, one easily verifies that—except for contrived examples—$R_\infty = \Sigma$, so that indeed $R_\infty^T \cdot R_\infty = X_\infty = \Sigma^2$.

The above *QR*-type algorithm is operated without shifts of the origin [11]. Therefore, convergence to the complete singular-value decomposition is likely to be very slow. On the other hand, with the zero shift this algorithm is particularly suitable for isolating a cluster of small (close to zero) singular values. This is typically the case when dealing with a tracking problem with a reasonable signal-to-noise ratio. Therefore, the above algorithm rapidly converges to the *URV* form, as given in the previous section, and thus may be called a "*URV* algorithm" too. Notice that if the signal-to-noise ratio is poor, convergence of this scheme may be poor, but one can then expect difficulties as well with the *URV* approach. For more details on how this algorithm relates to (the refinement step in) the *URV* algorithm of [21], we refer to [15] and to Section VII.

IV. JACOBI *QR*

The next step is to turn the above algorithm into a Jacobi-type algorithm, based on 2×2 transformations. First we add a permutation matrix in the

[2] L_k instead of $R_{k\sharp}^T$, and R_{k+1} instead of $R_{k\dagger}$.

above algorithm, given as [3]

$$\Pi = \begin{bmatrix} 0 & & & & 1 \\ & & & 1 & \\ & & \iddots & & \\ & 1 & & & \\ 1 & & & & 0 \end{bmatrix}.$$

The algorithm then works with upper triangular matrices only:

$$R_0 \Leftarrow R_A;$$

FOR $k = 0, \ldots, \infty$

$$\begin{bmatrix} \underbrace{\Pi L_k \Pi}_{\text{upper}} \Leftarrow \Pi \cdot \underbrace{R_k}_{\text{upper}} \cdot \underbrace{Q_{k\sharp}\Pi}_{\text{orthogonal}} \\ \underbrace{R_{k+1}}_{\text{upper}} \Leftarrow \underbrace{Q_{k\dagger}^T \Pi}_{\text{orthogonal}} \cdot \underbrace{\Pi L_k \Pi}_{\text{upper}} \cdot \Pi \end{bmatrix}$$

END

The first part may be viewed as applying a row permutation Π, and meanwhile preserving the upper triangular structure by applying orthogonal column transformations. Similarly, the second part may be viewed as applying a column permutation Π, and meanwhile preserving the upper triangular structure by applying orthogonal row transformations. To obtain a Jacobi-type algorithm, it suffices to split up these transformations into a sequence of 2×2 transformations. We consider the case here where m is even.[4] It is well known that Π can be split up with a so-called "odd-even ordering" as follows:

$$\Pi = \left(\underbrace{\Pi_{1|2}\Pi_{3|4} \cdots \Pi_{m-1|m}}_{\text{"odd"}} \cdot \underbrace{\Pi_{2|3}\Pi_{4|5} \cdots \Pi_{m-2|m-1}}_{\text{"even"}} \right)^{m/2},$$

where $\Pi_{i|i+1}$ differs from an identity matrix only in the entries $(i, i) = (i+1, i+1) = 0$ and $(i+1, i) = (i, i+1) = 1$. Each time a row (column)

[3] Note that $\Pi^T = \Pi$.

[4] Similar formulas apply for the case where m is odd.

permutation $\Pi_{i|i+1}$ is applied, a corresponding column (row) transformation restores the upper triangular structure. This is very similar to inserting appropriate permutations in the Kogbetlientz algorithm for computing the SVD of a triangular matrix [13] in order to maintain the upper triangular form at each step. With this, we finally obtain the Jacobi-type square-root *QR* algorithm (with odd-even ordering):

$$R \Leftarrow R_A,$$

$$\bar{U} \Leftarrow I,$$

$$\bar{V} \Leftarrow I;$$

FOR $k = 0, \ldots, \infty$
  FOR $j = 1, \ldots, \mathrm{m}/2$
    FOR $i = \underbrace{1, 3, \ldots, m-1}_{\text{odd}}, \underbrace{2, 4, \ldots, m-2}_{\text{even}}$

$$\begin{cases} R \Leftarrow \Pi_{i|i+1} \cdot R \cdot \bar{V}_{[i,j,k]} \\ \bar{U} \Leftarrow \bar{U} \cdot \Pi_{i|i+1} \\ \bar{V} \Leftarrow \bar{V} \cdot \bar{V}_{[i,j,k]} \end{cases}$$

    END
  END
  FOR $j = 1, \ldots, m/2$
    FOR $i = \underbrace{1, 3, \ldots, m-1}_{\text{odd}}, \underbrace{2, 4, \ldots, m-2}_{\text{even}}$

$$\begin{cases} R \Leftarrow \bar{U}^T_{[i,j,k]} \cdot R \cdot \Pi_{i|i+1} \\ \bar{U} \Leftarrow \bar{U} \cdot \bar{U}_{[i,j,k]} \\ \bar{V} \Leftarrow \bar{V} \cdot \Pi_{i|i+1} \end{cases}$$

    END
  END
END

Again, $\bar{U}_{[i,j,k]}$ and $\bar{V}_{[i,j,k]}$ represent orthogonal transformations in the $\{i, i+1\}$ plane, with rotation angles such that $\bar{U}_{[i,j,k]}$ or $\bar{V}_{[i,j,k]}$ restores the upper triangular form after the column or row permutation. At each stage we have

$$R_A = \bar{U} \cdot R \cdot \bar{V}^T,$$

and again, each iteration reduces the norm of the off-diagonal part in R. In other words, R converges to a diagonal matrix, resulting in the required SVD:

$$A = Q_A \cdot R_A$$

$$= \underbrace{Q_A \cdot \bar{U}}_{U} \cdot \underbrace{R}_{\Sigma} \cdot \underbrace{\bar{V}^T}_{V^T} .$$

V. PARALLEL AND ADAPTIVE SVD *URV* UPDATING

The above SVD-*URV* algorithm is also simple, and directly amenable to parallel implementation. The array of [13] may be used here; see Figure 1. Dots correspond to matrix entries; 2×2 frames may be thought of as processors. The triangular part stores the matrix R, while $\bar{V}$ is stored in the upper square part. Matrix $\bar{U}$ is not stored, as we will not need it in the adaptive case; see below. The "odd transformations" ($i = 1, 3, \ldots$) are computed in parallel on the main diagonal [Figure 1(a)] and then propagated to the blocks next outward (column transformations are propagated upwards, row transformations are propagated to the right). In Figure 1(c), this first set of transformations has moved far enough to allow the next set to be generated ("even transformations," this time). After Figure 1(d) comes Figure 1(a) again, etc. The only difference with [13] is that each 2×2 block now only performs either row or column transformations (plus permutations), instead of 2-sided transformations. The array will thus operate roughly twice as fast.

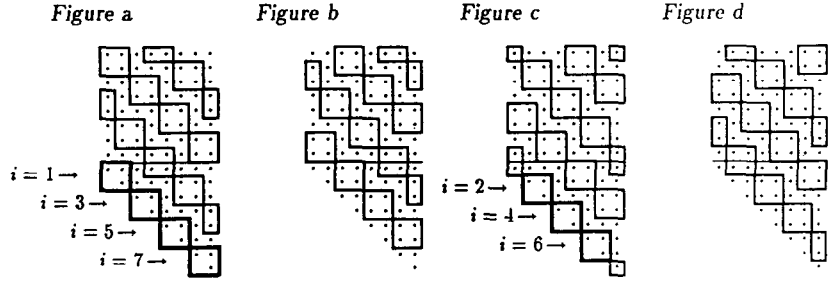

FIG. 1

Let us now return to the recursive case, with

$$A_k = \begin{bmatrix} \lambda A_{k-1} \\ \hline a_k^T \end{bmatrix},$$

and turn the algorithm of the previous section into an adaptive algorithm. In [17], it is shown how a Jacobi-type process may be interlaced with QR updates whenever new observations have to be worked in. The main difficulty of the systolic implementation is the fact that the two computational flows are in opposite directions: one flow is associated with the SVD-URV computations on a triangular matrix and updating the corresponding column transformation matrix in a square array, and another is associated with applying this transformation matrix to the new incoming observation vectors. The crux of the implementation is to patch up the flows as they cross each other. It is instructive to look at the systolic implementation first, and then derive the corresponding algorithmic description.

Figure 2 is similar to Figure 1, only the interlaced updates are added. The data vectors a_k are fed in into the upper $\bar{V}$-array in a skewed fashion, as indicated with the large ■'s, and propagated to the right, in between two transformation fronts (frames). The first step is to compute the matrix-vector product $\tilde{a}_k^T = a_k^T \cdot \bar{V}$, to put the new vector in the same "basis" as the current R-matrix. This is computed on the fly, with intermediate results being passed on upwards. The resulting vector $\tilde{a}_k$ becomes available at the top end of the square array, and is then reflected and propagated downwards, towards the triangular array, indicated with the small ■'s. While going downwards, the $\tilde{a}_k$-vector crosses upgoing transformations. These should then be applied to the $\tilde{a}_k$-vector too, in order to obtain consistent results. The QR updating is performed in the triangular part, much like in the conventional Gentleman-

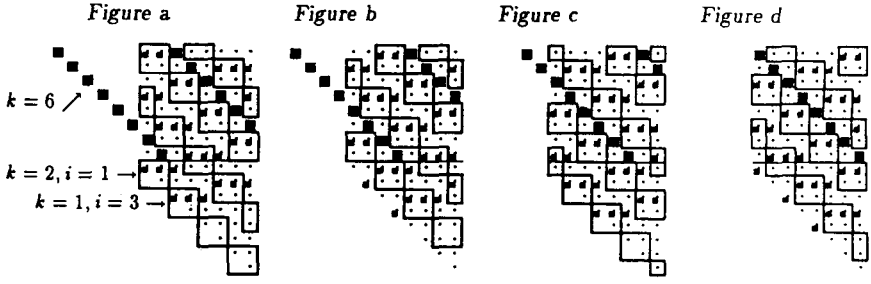

FIG. 2

Kung array [10], but the pipelining is somewhat different here (compatible with the Jacobi-type algorithm). Rotations are generated on the main diagonal and propagated to the right. In each 2×2 frame, column and row transformations corresponding to the SVD *URV* scheme are performed first, while in a second step, only row transformations are performed corresponding to the *QR* updating (affecting the $\tilde{a}_k$-components and the upper part of the 2×2 blocks). For further details concerning this array, we refer to [17].

An algorithmic description of this systolic implementation is given as follows:

$\bar{V} \Leftarrow I_{m \times m}$,
$R \Leftarrow O_{m \times m}$;

FOR $k = 1, \ldots, \infty$
input new observation vector a_k
1. Matrix-vector multiplication:

$$\tilde{a}_k^T \Leftarrow a_k^T \cdot \bar{V}$$

2. *QR* updating:

$$\begin{bmatrix} R \\ \hline 0 \end{bmatrix} \Leftarrow Q_k^T \cdot \begin{bmatrix} \lambda \cdot R \\ \hline \tilde{a}_k^T \end{bmatrix}$$

3. SVD/*URV* steps:
FOR $i = 1, \ldots, m-1$
IF $2k + i \pmod{2n} < n$
$R \Leftarrow \Pi_{i|i+1} \cdot R \cdot \bar{V}_{[i,k]}$
$\bar{V} \Leftarrow \bar{V} \cdot \bar{V}_{[i,k]}$
ELSE
$R \Leftarrow \bar{U}_{[i,k]}^T \cdot R \cdot \Pi_{i|i+1}$
$\bar{V} \Leftarrow \bar{V} \cdot \Pi_{i|i+1}$
END
END

Again, $\bar{U}_{[i,k]}$ and $\bar{V}_{[i,k]}$ represent orthogonal transformations in the $\{i, i+1\}$ plane, with rotation angles such that $\bar{U}_{[i,k]}$ or $\bar{V}_{[i,k]}$ restores the upper triangular form after the column or row permutation.

The backbone of the algorithm is the SVD-*URV* process. Whenever a new vector a_k has to be worked in, the process is interlaced with a *QR*

update (step 2) with $\tilde{a}_k^T = a_k^T \bar{V}$ (step 1). To have an algorithm with a fixed number of operations per loop, only one sequence of transformations for $i = 1, 2, \ldots, m-1$ is performed in step 3.[5] Here we make use of the well-known fact that an odd-even ordering can be reorganized into sequences of transformations, where in each sequence we have $i = 1, 2, \ldots, m-1$ (up to a different startup phase). One then only has to be careful with the computation of the transformations, i.e. the decision whether the permutation has to be applied to the left or to the right. This is done with the "if...then..." statement, which is explained as follows. In Figure 2(a) it is seen that the 2×2 blocks on the diagonal correspond to different updates, namely $k = 2$ and $k = 1$ (one could also add $k = 0$ and $k = -1$). The sum $2k + i$ is then indeed constant along the diagonal. In other words, the decision whether to perform a row permutation or a column permutation should only depend on $2k + i$. On the other hand, one should switch from row permutations to column permutations, or the other way around, after each $(n/2)$th update (for a fixed value of i).

The above algorithm has an $O(m^2)$ computational complexity per update (per loop). At each time, an approximate (*URV*-type) decomposition is available. The performance analysis of [16] straightforwardly carries over to this algorithm. This means that the tracking error (see Section II) is bounded by the time variation in m time steps; see [16] for details. The tracking experiments of [16] may be repeated here, yielding much the same results. Finally, with a systolic array with $O(m^2)$ processors (see Figure 2), an $O(m^0)$ throughput is achieved, which means that new vectors can be fed in at a rate which is independent of the problem size m.

VI. QSVD AND *QURV* UPDATING

The above updating algorithm/array is readily extended to generalized decompositions for matrix pairs, viz. the quotient singular-value decomposition (QSVD) or a similar *QURV*. Apart from the data matrix $A(N \times m)$, a second matrix $B(p \times m)$ is given, which for most applications corresponds to an error covariance matrix $B^T B$. The idea is then mostly to replace methods which are based on the SVD or *URV* decomposition of A, by methods which are based on decompositions of AR_B^{-1}, where R_B is, e.g., an $m \times m$ triangular factor of B ($B = Q_B R_B$). The postmultiplication with R_B^{-1} represents a pre-whitening operation. The key point is that AR_B^{-1} should never be computed explicitly, because both the inverse and the multiplication

[5] From then on, the algorithm will only provide an approximate decomposition. An exact diagonalization is only obtained with a possibly infinite number of SVD steps after each update.

may introduce additional errors, or R_B may not be invertible at all. The QSVD of the matrix pair $\{A, B\}$ or $\{A, R_B\}$, which is given by

$$A = Q_A \cdot \underbrace{U_A \cdot (\Sigma_A R) \cdot Q^T}_{R_A},$$

$$B = Q_B \cdot \underbrace{U_B \cdot (\Sigma_B R) \cdot Q^T}_{R_B},$$

reveals the SVD of AR_B^{-1} in an implicit way. Here Σ_A and Σ_B are diagonal matrices, R is upper triangular, and $U_A^T U_A = U_B^T U_B = Q^T Q = I$. For details, the reader is referred to [20].

Starting from the square triangular factors R_A and R_B, the QSVD may be computed with an iterative procedure, similar to the SVD procedure:

$R_1 \Leftarrow R_A$,
$R_2 \Leftarrow R_B$,
$\overline{U}_A \Leftarrow I$,
$\overline{U}_B \Leftarrow I$,
$\overline{Q} \Leftarrow I$;

FOR $k = 1, \ldots, \infty$
  FOR $i = 1, \ldots, m-1$
    $R_1 \Leftarrow \overline{U}_{A[i,k]}^T \cdot R_1 \cdot \overline{Q}_{[i,k]}$
    $R_2 \Leftarrow \overline{U}_{B[i,k]}^T \cdot R_2 \cdot \overline{Q}_{[i,k]}$
    $\overline{U}_A \Leftarrow \overline{U}_A \cdot \overline{U}_{A[i,k]}$
    $\overline{U}_B \Leftarrow \overline{U}_B \cdot \overline{U}_{B[i,k]}$
    $\overline{Q} \Leftarrow \overline{Q} \cdot \overline{Q}_{[i,k]}$
  END
END

The matrices R_1 and R_2 then converge to $\Sigma_A R$ and $\Sigma_B R$. The matrices $\overline{U}_{A[i,k]}$, $\overline{U}_{B[i,k]}$, and $\overline{Q}_{[i,k]}$ again represent orthogonal transformations in the $\{i, i+1\}$ plane. These transformations correspond to a 2×2 QSVD of $[R_1]_{i,i+1}$ and $[R_2]_{i,i+1}$, i.e. the submatrices on the intersections of rows $i, i+1$ and columns $i, i+1$ in R_1 and R_2 [14]. The key point is that (if R_2 is invertible)

$$[R_1]_{i,i+1} \cdot [R_2]_{i,i+1}^{-1} = \left[R_1 \cdot R_2^{-1}\right]_{i,i+1}.$$

Computing the transformations in a numerically reliable way is a problem here; see e.g. [3, 2, 1]. Again, the above QSVD algorithm may be turned into

an adaptive and parallel updating algorithm, where new rows may be appended to either one or both of the matrices A and B [18].

Our aim is now to develop a *QR*-type QSVD algorithm, similar to what we had for the 1-matrix case. This is straightforward. The algorithm below is readily seen to be a square-root version of Algorithm 8.6-1 of [11] for the symmetric generalized eigenvalue problem:

$R_1 \Leftarrow R_A$,
$R_2 \Leftarrow R_B$,
$\overline{Q} \Leftarrow I$,
$\overline{U}_A \Leftarrow I$,
$\overline{U}_B \Leftarrow I$;

FOR $k = 0, \ldots, \infty$
  FOR $j = 1, \ldots, m/2$
    FOR $i = \underbrace{1, 3, \ldots, m-1}_{\text{odd}}, \underbrace{2, 4, \ldots, m-2}_{\text{even}}$

$$\left[\begin{array}{l} R_1 \Leftarrow \Pi_{i|i+1} \cdot R_1 \cdot \overline{Q}_{[i,j,k]} \\ R_2 \Leftarrow \overline{U}^T_{B[i,j,k]} \cdot R_2 \cdot \overline{Q}_{[i,j,k]} \\ \overline{Q} \Leftarrow \overline{Q} \cdot \overline{Q}_{[i,j,k]} \\ \overline{U}_A \Leftarrow \overline{U}_A \cdot \Pi_{i|i+1} \\ \overline{U}_B \Leftarrow \overline{U}_B \cdot \overline{U}_{B[i,j,k]} \end{array}\right.$$

    END
  END
  FOR $j = 1, \ldots, m/2$
    FOR $i = \underbrace{1, 3, \ldots, m-1}_{\text{odd}}, \underbrace{2, 4, \ldots, m-2}_{\text{even}}$

$$\left[\begin{array}{l} R_2 \Leftarrow \Pi_{i|i+1} \cdot R_2 \cdot \overline{Q}_{[i,j,k]} \\ R_1 \Leftarrow \overline{U}^T_{A[i,j,k]} \cdot R_1 \cdot \overline{Q}_{[i,j,k]} \\ \overline{Q} \Leftarrow \overline{Q} \cdot \overline{Q}_{[i,j,k]} \\ \overline{U}_A \Leftarrow \overline{U}_A \cdot \overline{U}_{A[i,j,k]} \\ \overline{U}_B \Leftarrow \overline{U}_B \cdot \Pi_{i|i+1} \end{array}\right.$$

    END
  END
END

Unlike in the first QSVD algorithm, computing the transformations is simple here. In the first loop, a row permutation is applied to R_1, and then $\overline{Q}_{[i,j,k]}$ is

computed to upper-triangularize R_1 again. Finally, $\overline{U}_{B[i,j,k]}$ is computed to upper-triangularize $R_2\overline{Q}_{[i,j,k]}$. The second loop starts with a permutation on R_2, etc. With the above algorithm, $R_1R_2^{-1}$ (if R_2 is invertible) converges to the *URV* form first, and then further on to diagonal form.

Finally, when the rows of A and B are both updated with time, an adaptive updating algorithm is straightforwardly obtained as follows. Only the orthogonal matrix $\overline{Q}$ is stored now:

$$\overline{Q} \Leftarrow I_{m\times m},$$
$$R_1 \Leftarrow O_{m\times m},$$
$$R_2 \Leftarrow O_{m\times m};$$

FOR $k = 1, \ldots, \infty$

input new observation vectors a_k, b_k

1. Matrix-vector multiplication:

$$\tilde{a}_k^T \Leftarrow a_k^T \cdot \overline{Q}$$
$$\tilde{b}_k^T \Leftarrow b_k^T \cdot \overline{Q}$$

2. QR updating:

$$\begin{bmatrix} R_1 \\ \hline 0 \end{bmatrix} \Leftarrow Q_{1_k}^T \cdot \begin{bmatrix} \lambda \cdot R_1 \\ \hline \tilde{a}_k^T \end{bmatrix}$$

$$\begin{bmatrix} R_2 \\ \hline 0 \end{bmatrix} \Leftarrow Q_{2_k}^T \cdot \begin{bmatrix} \lambda \cdot R_2 \\ \hline \tilde{b}_k^T \end{bmatrix}$$

3. SVD/*URV* steps:

FOR $i = 1, \ldots, m-1$

IF $2k + i \pmod{2n} < n$

$$R_1 \Leftarrow \Pi_{i|i+1} \cdot R_1 \cdot \overline{Q}_{[i,k]}$$
$$R_2 \Leftarrow \overline{U}_{Bi|i+1}^T \cdot R_2 \cdot \overline{Q}_{[i,k]}$$
$$\overline{Q} \Leftarrow \overline{Q} \cdot \overline{Q}_{[i,k]}$$

ELSE

$$R_2 \Leftarrow \Pi_{i|i+1} \cdot R_2 \cdot \overline{Q}_{[i,k]}$$
$$R_1 \Leftarrow \overline{U}_{Ai|i+1}^T \cdot R_1 \cdot \overline{Q}_{[i,k]}$$
$$\overline{Q} \Leftarrow \overline{Q} \cdot \overline{Q}_{[i,k]}$$

END

END

The corresponding systolic array is again given in Figure 2. The square part stores and updates $\underline{Q}$, while the triangular part stores and updates R_1 and R_2, overlaid. The large ■'s also correspond to overlaid data vectors a_k and b_k. The rest is similar to the one-matrix case.

In some applications one has that only A is updated, while B remains constant (but not the identity). It is clear that this is just a special case of the above scheme. One trivially skips the QR updating (with windowing) of R_B, and one obtains still an efficient implementation of this simpler case.

VII. RELATION TO OTHER WORK

The decompositions discussed in this paper and related references [4, 7, 15, 16, 21, 22] all compute an "approximate decomposition" of a matrix A with a triangular matrix in the middle as follows:

$$A = \left[\tilde{U}_s \mid \tilde{U}_n\right] \cdot \left[\begin{array}{c|c} R_s & R_{sn} \\ \hline & R_n \end{array}\right] \cdot \left[\begin{array}{c} \tilde{V}_s^T \\ \hline \tilde{V}_n^T \end{array}\right],$$

whereby $\sigma_{\max}(R_n) = \eta$ and $\sigma_{\max}(R_{sn}) = \varepsilon$ are small, and $\sigma_{\min}(R_s) = \delta$ is reasonably larger than η and ε. As a result, the singular values of R_s approximate well the large singular values of A. Those of R_n are good approximations of the small singular values of A only if $\varepsilon \ll \eta$. The desired ordering is thus $\delta > \eta \gg \varepsilon$.

(1) In [16], this was called an "approximate SVD." When one or more sweeps of a Kogbetliantz-type SVD algorithm are applied to the triangular array, quadratic convergence was observed for the off-diagonal part R_{sn} even when the required adjacency of close singular values was not respected. With one SVD sweep one reduces the norm of R_{sn} from ε to $\varepsilon^2/(\delta - \eta)$.

(2) In [22] a "URV decomposition" approach was proposed, based on estimating small singular values of a matrix A and deflating them to the R_n-block (hence ordering is obtained). An adaptive version of this for matrices A_k was then proposed in [21]. Finally, a refinement idea for such URV decompositions was proposed and analysed in [7, 22]. One refinement step essentially amounts to a QR decomposition and has the similar effect of reducing the norm of R_{sn} from ε to $\varepsilon_2 \doteq \varepsilon^2(\delta - \eta)$, but also flips the matrix around to a (block) lower triangular one. Therefore [22] recommends a second step to flip it over again to upper triangular form and hence reduce the norm of R_{sn} further to $\varepsilon_2^2/(\delta - \eta) = \varepsilon^4/(\delta - \eta)^3$. Notice that the number of operations of one SVD sweep is roughly equal to that of two QR refinement steps. An analysis of the related QR flipping is also given in [5].

(3) In this paper we show that similar refinement steps can in fact be performed while preserving the upper triangular form at no extra cost. So, instead of having an improved decomposition at the same cost of an SVD sweep, we propose a cheaper procedure with the same refinement property as an SVD sweep. Moreover, the parallel implementation fits nicely on the same array as the SVD updating scheme. Another difference with the *URV* approach is that no rank test is really needed at any stage. This follows the philosophy of the GSVD algorithm of Paige [19] (in contrast with earlier versions such as e.g. [20]), where rank decisions are deferred if possible. Of course, when identifying the noise subspace a rank decision is required, but it is not needed in the recursive update of the decomposition. Therefore rank decisions at each step are independent of each other, which is not the case in the *URV* approach. We rely here on the self-ordering property of the *QR* algorithm to expect that in the adaptive case smaller singular values will automatically cluster. This of course can only be expected if the noise subspace corresponding to these small singular values does not change too much with each time step k.

Extensions to updating the implicit decomposition for two matrices A and B can be found in [18] (GSVD) and [4] (*GURV*). The approach chosen by [4] is to extend the work of [22] to the case of an implicit decomposition. This first requires a joint *QR* decomposition of the matrices A and B in order to extract the common null space and triangular factor R_{AB}. The resulting matrices Q_A and Q_B then have a joint decomposition known as the *CS* decomposition, which can be updated adaptively as new observations are being collected [4]. Two weaknesses of this approach are the preliminary rank determination of the joint *QR* factorization of A and B, and the two-stage updating required when new observations are being processed. The main advantage of the method is that the rank revealing part of the *QURV* decomposition is now concentrated in the matrix Q_A. In the present paper we focused on the parallel implementation of the *QURV* decomposition and chose for this reason not to perform the preliminary extraction of the triangular factor R_{AB}. The resulting algorithm is again easy to implement on the SVD array of [17] and does not require the double flipping of the *URV* as in [22]. The algorithm is then very close to the one presented for the single-matrix case, and earlier remarks apply here again when the matrix B is not too badly conditioned.

This research was partially sponsored by ESPRIT Basic Research action 3280. Marc Moonen is a senior research assistant with the Belgian N.F.W.O. (National Fund for Scientific Research). Paul Van Dooren was partially supported by the Research Board of the University of Illinois at Urbana-

Champaign (grant P 1-2-68114) and by the National Science Foundation (grant CCR 9209349). Filiep Vanpoucke is a research assistant with the Belgian N.F.W.O.

Part of this research was done while the first two authors were visiting the Institute of Mathematics and Applications, Minneapolis. The Institute's financial support is gratefully acknowledged.

REFERENCES

1 G. E. Adams, A. W. Bojanczyk, and F. T. Luk, Computing the PSVD of two 2×2 triangular matrices, *SIAM J. Matrix Anal. Appl*, to appear.

2 Z. Bai and J. Demmel, Computing the Generalized Singular Value Decomposition, Report UCB/CSD 91/645, Computer Science Div., Univ. of California, Berkeley, Aug. 1991; *SIAM J. Sci. Statist. Comput.*, submitted for publication.

3 A. W. Bojanczyk, L. M. Ewerbring, F. T. Luk, and P. Van Dooren, An accurate product SVD algorithm, *Signal Process.* 25:189–202 (1991).

4 A. Bojanczyk, F. T. Luk, and S. Qiao, Generalizing the *URV* decomposition for adaptive signal parameter estimation, in *Proceedings of the IMA Workshop on Linear Algebra in Signal Processing, April 6–10, 1992, Minneapolis, Minnesota,* to appear.

5 S. Chandrasekaran and I. Ipsen, Analysis of a *QR* Algorithm for Computing Singular Values, Res. Rept. YALEU/DCS/RR-917, Dept. of Computer Science, Yale Univ., Aug. 1992.

6 J.-M. Delosme and I. Ipsen, From Bareiss' algorithm to the stable computation of partial correlations, *J. Comput. Appl. Math.* 27:53–91 (1989).

7 E. M. Dowling, L. P. Ammann, and R. D. DeGroat, A *TQR*-Iteration Based Adaptive SVD for Real Time Angle and Frequency Tracking, Technical Report, Erik Jonsson School of Engineering and Computer Science, Univ. of Texas at Dallas.

8 V. Fernando, Linear convergence of the row cyclic Jacobi and Kogbetliantz methods, *Numer. Math.* 56:71–91 (1989).

9 V. Fernando and B. Parlett, Accurate Singular Values and Differential *QD* Algorithms, Report UCB/PAM-554, Center for Pure and Applied Mathematics, Univ. of California, Berkeley, July 1992.

10 W. M. Gentleman and H. T. Kung, Matrix triangularization by systolic arrays in *Real-Time Signal Processing IV,* Proc. SPIE 298, 1982, pp. 19–26.

11 G. H. Golub and C. F. Van Loan, *Matrix Computations,* North Oxford Academic, Johns Hopkins U.P., 1988.

12 E. Kogbetliantz, Solution of linear equations by diagonalization of coefficient matrices, *Quart. Appl. Math.* 13:123–132 (1955).

13 F. T. Luk, A triangular processor array for computing singular values, *Linear Algebra Appl.* 77:259–273 (1986).

14 F. T. Luk, A parallel method for computing the GSVD, *Internat. J. Parallel Distributed Comput.* 2:250–260 (1985).

15 R. Mathias and G. W. Stewart, A Block *QR* Algorithm and the Singular Value Decomposition, Technical Report CS-TR 2626, Dept. of Computer Science, Univ. of Maryland, 1992.
16 M. Moonen, P. Van Dooren, and J. Vandewalle, An SVD Updating Algorithm for Subspace Tracking, *SIAM J. Matrix Anal. Appl* 13:1015–1038 (1992).
17 M. Moonen, P. Van Dooren, and J. Vandewalle, A Systolic Array for SVD Updating, *SIAM J. Matrix Anal. Appl.*, 14:353–371 (1993).
18 M. Moonen, P. Van Dooren, and J. Vandewalle, A systolic algorithm for QSVD updating, *Signal Process.* 25:203–213 (1991).
19 C. C. Paige, Computing the generalized singular value decomposition, *SIAM J. Sci. Statist. Comput.* 7:1126–1146 (1986).
20 C. C. Paige and M. Saunders, Towards a generalized singular value decomposition, *SIAM J. Numer. Anal.* 18:398–405 (1981).
21 G. W. Stewart, An Updating Algorithm for Subspace Tracking, Technical Report CS-TR 2494, Dept. of Computer Science, Univ. of Maryland, 1990; *IEEE Trans. Signal Process.*, to appear.
22 G. W. Stewart, On an Algorithm for Refining a Rank-Revealing *URV* Decomposition, Technical Report CS-TR 2626, Dept. of Computer Science, Univ. of Maryland, 1991; *Linear Algebra Appl.*, to appear.

Received 1 August 1992; final manuscript accepted 13 February 1993